Randomized Algorithms in Deep Learning

Torumoy Ghoshal
PhD Student
Department of Computer and Information Science
University of Mississippi

November 30, 2017

Introduction

It has been shown that randomization based training methods can significantly boost the performance or efficiency of neural networks.[1] Among these methods, most approaches use randomization either to change the data distributions, and/or to fix a part of the parameters or network configurations.[1]

The power of neural networks comes from the nonlinear function in the hidden units used to model the nonlinear mapping between input and output. Unfortunately, this kind of architecture loses the elegance of finding the global minimum solution with respect to all the parameters of the network since the loss function depends on the output of nonlinear neurons. Thus, the optimization turns out to be nonlinear least square problem which is usually solved iteratively[1]. In this case, the error function has to be back propagated backwards to serve as a guidance for tuning the parameters[2]. This results in a very slow network which may not converge. Randomization based methods solve this problem by randomly fixing the network configurations. A more robust system can also be achieved by randomly corrupting the input data or parameters during training. These approaches have lead to remarkable results even when a single hidden layer feed forward network was used.

Some deep networks where randomization is heavily used are perception, standard feed-forward neural network with randomization, random vector functional neural network, radial basis function network, recurrent neural network, convolutional neural network and few others. The benefits that these Deep Neural Networks get from using randomization are that they are faster, more robust, easy to analyze and can exploit modern hardware architectures.

In my project, I explored some effects of randomization in convolutional neural networks. More specifically, I observed the effects of batch size variation and Dropout in Convolutional Neural Network (CNN).

Stochastic Gradient Descent

Stochastic gradient descent (often shortened to SGD), also known as incremental gradient descent, is a stochastic approximation of the gradient descent optimization and iterative method for minimizing an objective function that is written as a sum of differentiable functions. In other words, SGD tries to find minima or maxima by iteration[3].

Opposed to SGD, the cost function for Batch Gradient Descent (BGD) algorithm is given by:

$$J_{train}\theta = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - (y^{(i)}))^2$$

And the gradient update rule is given by:

$$\text{Repeat}\{$$

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - (y^{(i)}))x_j^i$$

$$\text{for every j = 0...n } \}$$

For the algorithm above, after computing for all values of m, the gradient descent advances only 1 step. For large m, the memory requirement is also very high. Hence, not only that the algorithm will be very slow, but it might become quite expensive to implement.

The SGD cost function is given by:

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{train}\theta = \frac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$

And the update rule:

$$\text{Repeat } \{ \text{ for i = 1...m } \{$$

$$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - (y^{(i)}))x_j^i$$

$$\text{for j=0...n } \} \ \}$$

The absence of the $\sum$ term in the update rule helps SGD to advance for each training example, which is significantly faster compared to BGD. This also benefits the memory requirement. In practice, SGD is performed in multiple passes over the training set and the algorithm converges much faster compared to BGD.

Dropout

Dropout is a technique to address over-fitting in very large neural networks. "During training, dropout samples from an exponential number of different thinned networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods"[5].

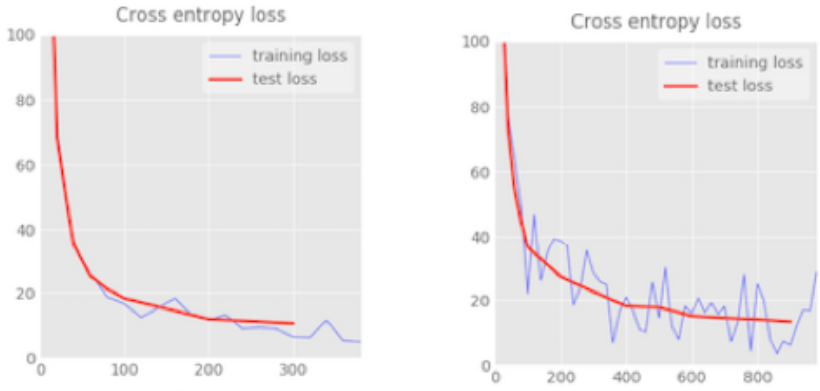
In practice, in case of CNN, and also in my project, some random nodes are selected at the training step and their value is set to 0.

Experiment Setup

Experiments were run on MNIST dataset with 50000 training samples and 10000 test samples. Tools included Google' Tensorflow. Effects of variation of batch size and dropout were observed from an implementation of CNN on the MNIST dataset. RNN was implemented on the MNIST dataset.

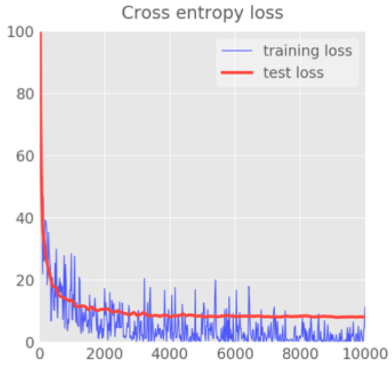
Results

Effect of batch size:



Above are two screenshots taken at the 30th second after CNN starts running. This implementation uses ADAM optimizer. However, the effect of batch size variation should be similar to SGD.The left one corresponds to batch size = 1000 and the right batch size = 100. It is seen that the right completes approximately 1000 iterations in 30 seconds, and the left only 400. Though there is jitter in the training loss for the right one, after 10000 iterations the jitter minimizes and the system gets very close to global minima. The left one also does so, but it is significantly slower.

Effect of dropout:



Above are two screenshots that represent the effect of "no dropout" and "dropout." In red lines represent accuracy. In the left one, without dropout, there is no sign that accuracy will increase with more iterations. In fact the the slope is upwards. In the right one, with dropout, we get a downward slope and better accuracy.

RNN: It is well known that CNN performs better than other deep learning algorithms for image data. I implemented RNN on the MNIST dataset and the accuracy was approximately 89%. This is quite low compared to $\geq 98\%$ acieved by CNN.

References:

1. Le Zhang, P.N. Suganthan, "A survey of randomized algorithms for training neural networks", Information Sciences 364-365 (2016), 146-155

2. S. Haykin , N. Network , Neural Networks: A comprehensive foundation, Neural Netw. 2 (2004) (2004).

3. https://en.wikipedia.org/wiki/Stochastic_gradient_descent

4. https://www.coursera.org/learn/machine-learning/lecture/DoRHJ/stochastic-gradient-descent

5. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Journal of Machine Learning Research 15 (2014) 1929-1958